Überblick

REST & Virtualisierung
- RESTful Web-Services
  - Einführung
  - Implementierung mittels JAXB
- Virtualisierung
  - Einführung
  - Aufbau einer virtuellen Maschine
  - Erstellen einer virtuellen Maschine
  - Zusammenfassung und Ausblick
- Aufgabe 2

Representational State Transfer (REST)

- REST
  - HTTP als Anwendungsprotokoll
    - `PUT`-Methode zum Anlegen einer Ressource
    - `GET`-Methode zum Auslesen einer Ressource
    - ...
  - Direkte Adressierung der Ressourcen
- Beispiel: Dienst zur Verwaltung mehrerer Drucker
  - Dienst-URL: `http://localhost:12345/printer-service/`
  - Adressierung eines Druckers über eigene URL, z. B. `http://localhost:12345/printer-service/printer0`
  - Client-Methode

```
public String print(String printer, String text);
```

- Java Architecture for XML Binding (JAXB)
  - Standardmäßig integriert in Java
  - Erzeugung von Java-Klassen aus einem XML-Schema

XML Schema — Benutzerdefinierte Datentypen

- Definition eigener einfacher Datentypen (`simpleType`)

```
<xsd:simpleType name="[Name des Datentyps]">
  [Beschreibung des Datentyps]
</xsd:simpleType>
```

  - Liste

```
<xsd:list itemType="[Datentyp der Listenelemente]"/>
```

  - Union: Datentyp mit kombiniertem Wertebereich

```
<xsd:union memberTypes="[Aufzaehlung erlaubter Datentypen]"/>
```

- Ableitung eines Basisdatentyps mit Einschränkung des Wertebereichs

```
<xsd:restriction base="[Basisdatentyp]">
  [Einschraenkung des Basisdatentyps]
</xsd:restriction>
```

  - Aufzählung: Festlegung bestimmter zulässiger Werte (`enumeration`)
  - Minimal-/Maximalwerte für Integer (`minInclusive`, `maxInclusive`)
  - Reguläre Ausdrücke für Zeichenketten (`pattern`)
  - ...

XML Schema — Benutzerdefinierte Datentypen

- Definition eigener komplexer Datentypen (`complexType`)

```
<xsd:complexType name="[Name des Datentyps]">
  [Beschreibung des Datentyps]
</xsd:complexType>
```

  - Festlegung der Reihenfolge von Subelementen

```
<xsd:sequence>[Subelemente]</xsd:sequence>
```

  - Referenzierung von existierenden Datentypen

```
<xsd:element ref="[Name des Datentyps]"/>
```

- Spezifizierung eigener Elementnamen und Zuordnung zu Datentypen

```
<xsd:element name="[Elementname]" type="[Name des Datentyps]"/>
```

- Definition eigener Attribute (nur einfache Datentypen erlaubt)

```
<xsd:attribute name="[Attributname]" type="[Name des Datentyps]"/>
```

- Attribute zur Einschränkung der Anzahl von Elementen
  - Festlegung einer Mindest- bzw. Maximalanzahl (`minOccurs`, `maxOccurs`)
  - Für optionale Elemente: `minOccurs` auf `0` setzen

Definition der Datentypen und Nachrichten

- Vordefinierte Datentypen (Beispiele)
  - `xsd:boolean`, `xsd:int`, `xsd:long`, `xsd:string`
  - `xsd:time`, `xsd:date`
- Komplexe Datenstrukturen, z. B. `Boolean`-Liste

```
<xsd:element name="list" type="xsd:boolean"
             minOccurs="0" maxOccurs="unbounded"/>
```

- Beispiel (`printer.xsd`)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- Data Types -->
    <xsd:complexType name="MWText">
        <xsd:sequence>
            <xsd:element name="text" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>

    <!-- Messages -->
    <xsd:element name="MWPrinterRequest" type="MWText"/>
    <xsd:element name="MWPrinterReply" type="MWText"/>
</xsd:schema>
```

Erzeugung von Hilfsklassen

- Verwendung des Binding-Compiler `xjc`
- Beispielaufruf

```
$ xjc -p mw.printer.generated -d src printer.xsd
```

Annahmen: Zu erzeugendes Package ist `mw.printer.generated`, Zielordner für erzeugte Dateien ist `src`

- Erzeugte Klassen
  - Eine Klasse für jeden spezifizierten Datentyp

```
public class MWText {
    protected String text;
    public String getText() { return text; }
    public void setText(String value) { this.text = value; }
}
```

  - `ObjectFactory` zur Instanziierung von Datentypen und Nachrichten

```
public class ObjectFactory {
    public MWText createMWText() { return new MWText(); }
    public JAXBElement<MWText>
             createMWPrinterRequest(MWText value) { [...] }
    public JAXBElement<MWText>
             createMWPrinterReply(MWText value) { [...] }
}
```

Implementierung der Client-Seite

Dienstabstraktion (`javax.xml.ws.Service`)

- Stellvertreterobjekt für entfernten Dienst: `Service`
  - Konfiguration der Verbindungsparameter
    - Dienst- bzw. Ressourcen-Adressierung
    - Kommunikationsprotokoll (**HTTP**, SOAP)
  - Factory für Objekte zum Dienstzugriff (siehe nächste Folie)
- Beispiel

```
public String print(String printer, String text) {
    // Zusammenstellung der Ressourcen-Adresse
    String path = "http://localhost:12345/printer-service/" +
                  printer;

    // Konfiguration der Service-Verbindung
    QName qName = new QName("", ""); // -> kein Endpunktname
    Service service = Service.create(qName);
    service.addPort(qName, HTTPBinding.HTTP_BINDING, path);

    [...] // siehe naechste Folien
}
```

Implementierung der Client-Seite

Dienstzugriff (`javax.xml.ws.Dispatch`)

- Schnittstelle zum Absetzen dynamischer Aufrufe: `Dispatch`
  - Spezifizierung des Zugriffs auf Nachrichten (`Service.MODE`)
    - `MESSAGE`: Zugriff auf vollständige Nachrichten
    - `PAYLOAD`: Zugriff auf Nachrichten-Payloads
  - Festlegung der HTTP-Methode
- Binding-Kontext (`javax.xml.bind.JAXBContext`): Informationen über Art und Zusammensetzung von Datentypen und Nachrichten
- Beispiel

```
// Erzeugung des Binding-Context
String contextPath = "mw.printer.generated";
JAXBContext jc = JAXBContext.newInstance(contextPath);

// Erzeugung des Dispatch
Dispatch<Object> dispatch = service.createDispatch(qName, jc,
                                        Service.Mode.PAYLOAD);

// Festlegung der HTTP-Methode
Map<String, Object> rc = dispatch.getRequestContext();
rc.put(MessageContext.HTTP_REQUEST_METHOD, "POST");
```

Implementierung der Client-Seite

Zusammenstellung der Anfrage

- Aufrufparameter
  - Erzeugung per `ObjectFactory`
  - Setzen der Attributwerte
- Anfragenachricht
  - Erzeugung per `ObjectFactory`
  - Kein eigener Datentyp, sondern generisches `JAXBElement`
- Beispiel

```
// Erzeugung der Objekt-Factory
ObjectFactory f = new ObjectFactory();

// Erzeugung des Aufrufparameters
MWText input = f.createMWText();
input.setText(text); // text: zu druckende Zeichenkette

// Erzeugung der Anfrage
JAXBElement<MWText> request = f.createMWPrinterRequest(input);
```

Implementierung der Client-Seite

Dienstaufruf und Auswertung der Antwort

- Aufrufvarianten von `Dispatch`
  - `invoke()`: Synchroner Aufruf mit Antwort (→ *Request-Reply*)
  - `invokeAsync()`: Asynchroner Aufruf mit Antwort
  - `invokeOneWay()`: Absetzen einer Anfrage (keine Antwort)
- Antwortnachricht
  - Gekapselt in `JAXBElement` (vgl. Anfragenachricht)
  - Auspacken des Rückgabewerts
- Beispiel

```
// Senden der Anfrage und Empfang der Antwort
JAXBElement reply = (JAXBElement) dispatch.invoke(request);

// Auswertung der Antwort
MWText status = (MWText) reply.getValue();
return status.getText();
```

Implementierung der Server-Seite

Dienstimplementierung (`javax.xml.ws.Provider`)

- Dienstendpunkt: `Provider`
  - `@WebServiceProvider`: Kennzeichnung eines öffentlichen Endpunkts
  - Spezifizierung des Zugriffs auf Nachrichten (hier: `PAYLOAD`, vgl. Client)
  - Aufruf der `invoke`-Methode für jede eintreffende Anfrage
  - Kapselung der Anfrage- und Antwortnachrichten in `Source`-Objekten

```
@WebServiceProvider
@ServiceMode(value=Service.Mode.PAYLOAD)
public class MWPrinterService implements Provider<Source> {
    public Source invoke(Source source) {
        [...] // siehe naechste Folien
    }
}
```

- Erzeugung und Veröffentlichung des Dienstendpunkts

```
Endpoint endpoint = Endpoint.create(HTTPBinding.HTTP_BINDING,
                                    new MWPrinterService());
endpoint.publish("http://localhost:12345/printer-service/");
```

Implementierung der Server-Seite

Zugriff auf den Anfragenkontext

- Web-Service-Kontext
  - Referenz auf die Web-Service-Umgebung
  - Initialisierung (Beispiel)
    - Definition einer (zunächst leeren) Referenz `wsContext`

```
@javax.annotation.Resource(type=WebServiceContext.class)
protected WebServiceContext wsContext;
```

    - `wsContext` wird beim Anlegen des Objekts von der Umgebung initialisiert
- Anfragenkontext
  - Wird für jeden Aufruf von `invoke()` aktualisiert
  - Zugriff auf die HTTP-Header der Anfrage
- Beispiel: Auslesen der HTTP-Methode der Anfrage sowie des Pfads der Ressource (→ Drucker), an die diese Anfrage gestellt wurde

```
MessageContext mc = wsContext.getMessageContext();
String httpMethod = (String) mc.get(MessageContext.HTTP_REQUEST_METHOD);
String path = (String) mc.get(MessageContext.PATH_INFO);
System.out.println(httpMethod + " request, printer " + path);
```

Implementierung der Server-Seite

Auspacken der Aufrufparameter

- Anfragenachricht
  - Bereitstellung eines `Unmarshaller` durch den Binding-Kontext
  - Repräsentation als `JAXBElement`
  - Extraktion der Aufrufparameter
- Beispiel

```
// Erzeugung des Binding-Context
String contextPath = "mw.printer.generated";
JAXBContext jc = JAXBContext.newInstance(contextPath);

// Unmarshalling der Anfrage
Unmarshaller u = jc.createUnmarshaller();
JAXBElement request = (JAXBElement) u.unmarshal(source);

// Auspacken des Aufrufparameters
MWText input = (MWText) request.getValue();
String text = input.getText();

[...] // Bearbeitung der Anfrage
```

Implementierung der Server-Seite

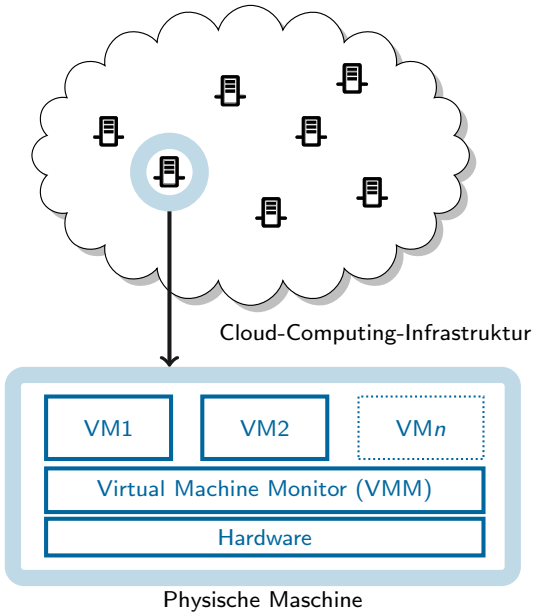
Zusammenstellung der Antwort

- Antwortnachricht
  - Vorgehen analog zur Zusammenstellung der Anfrage auf Client-Seite
  - Antwort als Rückgabewert der `invoke`-Methode
  - Kapselung der Antwort in einem `Source`-Objekt
- Beispiel

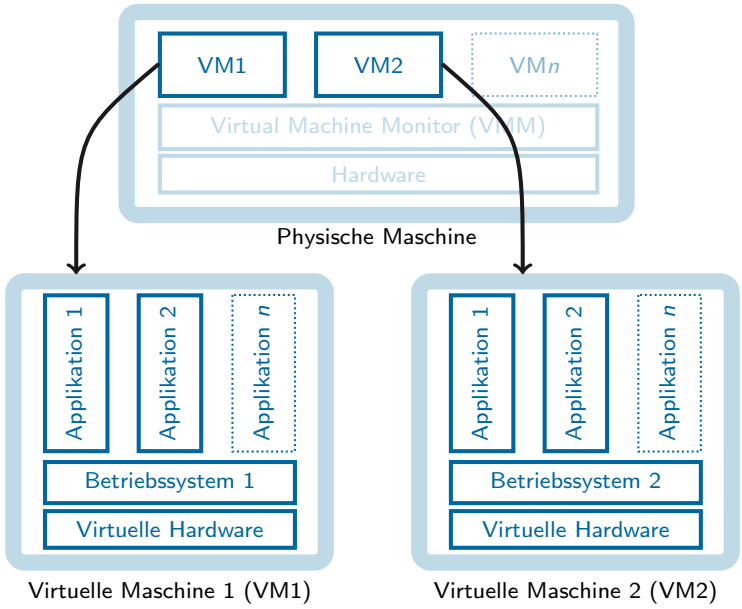
```
// Erzeugung der Objekt-Factory
ObjectFactory f = new ObjectFactory();

// Erzeugung des Rueckgabewerts
MWText status = f.createMWText();
status.setText("OK");

// Erzeugung der Antwort
JAXBElement<MWText> reply = f.createMWPrinterReply(status);

// Return aus der invoke()-Methode
Source replySource = new JAXBSource(jc, reply);
return replySource;
```

Überblick

Virtualisierung als Grundlage für Cloud Computing



Aufbau einer virtuellen Maschine



Aufbau einer virtuellen Maschine

- Notwendige Betriebsmittel
  - Physische Maschine und Gastgeberbetriebssystem („Host")
  - Virtualisierungssoftware, die den Virtual Machine Monitor bereitstellt
  - **Abbild der zu betreibenden virtuellen Maschine**
- Aufbau des Abbilds einer virtuellen Maschine
  - Meta-Informationen (spezifisch, je nach Virtualisierungssoftware)
  - **Dateisystem**, beinhaltet für gewöhnlich:
    - Kern des zu virtualisierenden Gastbetriebssystems („Guest")
    - User-Space-Komponenten des Gastbetriebssystems
    - Daten
- Analogie zur Objektorientierung
  - Das statische Abbild einer virtuellen Maschine entspricht einer **Klasse**
  - Eine im Betrieb befindliche virtuelle Maschine ist die **Instanz** eines solchen Abbilds

Entwicklung eines VM-Abbilds `dd(1)`

- Gebräuchliche Abbild-Typen für virtuelle Maschinen (VM)
  - Kopie eines Datenträgers (z. B. ISO-Image einer CD oder DVD):

```
$ dd if=/dev/sdb of=./cd-image.iso
$ file -b ./cd-image.iso
ISO 9660 CD-ROM filesystem data (bootable)
```

  - Erzeugen einer leeren Abbild-Datei:

```
$ truncate -s 100M image.raw
$ ls -lh image.raw
-rw-r--r-- 1 thoenig users 100M  4. Nov 12:11 image.raw
$ du image.raw
0
$ file -b image.raw
data
```

- Alternativ ist es möglich, einen physischen Datenträger als Basis für eine virtuelle Maschine zu verwenden

Entwicklung eines VM-Abbilds `qemu(1)`

- Die Erstellung und Aufbereitung des Abbilds der virtuellen Maschine benötigt erweiterte Privilegien (Root-Rechte)
- Die Aufbereitung des Abbilds geschieht daher isoliert in der Betriebsumgebung einer virtuellen Maschine („Live-System")
  - Linux-Live-System Grml (`http://grml.org`)
  - Emulator: `qemu` (`http://qemu.org`)
- Start des Live-Systems (Beispiel):

```
$ qemu -drive file=grml.iso,index=0,media=cdrom \
       -drive file=image.raw,index=1,media=disk
```

  - `root`-Dateisystem (Teil von `grml.iso`, Gerätepfad `/dev/sr0`) wird automatisch eingehängt, nicht jedoch das leere Abbild (`image.raw`, Gerätepfad `/dev/sda`)
  - Folgende Schritte sind innerhalb des Live-Systems durchzuführen!

Entwicklung eines VM-Abbilds `mkfs(8)`

- Um als Basis für eine virtuelle Maschine zu dienen, muss die Abbild-Datei `image.raw` ein Dateisystem beinhalten
- Das Kommando `mkfs` (**m**a**k**e **f**iles**y**stem) erzeugt Dateisysteme, der Parameter `-t` spezifiziert dabei den Dateisystemtyp
- Erstellen eines ext3-Dateisystems mit der Bezeichnung „`VM-Abbild`" auf dem blockorientierten Gerät (block device) `/dev/sda`:

```
$ mkfs -t ext3 -L "VM-Abbild" /dev/sda
mke2fs 1.41.11 (14-Mar-2010)
/dev/sda is entire device, not just one partition!
Proceed anyway? (y,n)
Filesystem label=VM-Abbild
OS type: Linux
```

Entwicklung eines VM-Abbilds `mount(8)`, `debootstrap(8)`

Einhängen, Bootstrapping

- Installation der User-Space-Komponenten des zukünftigen Gastbetriebssystems in das neu erzeugte, leere Dateisystem:

1. Einhängen des zuvor erstellten Dateisystems mit `mount`:

```
$ mount /dev/sda /mnt
```

   Kontrolle:

```
$ mount | grep sda
```

2. Erstellung der User-Space-Komponenten des Zielsystems mit `debootstrap`:

```
$ debootstrap --arch i386 wheezy /mnt/ 'http://debian.cs.fau.de/debian'
```

   Kontrolle:

```
$ ls -alR /mnt | more
```

3. Wechsel in das von `debootstrap` erstellte System mittels `chroot(8)`

```
$ chroot /mnt /bin/bash
```

Entwicklung eines VM-Abbilds `chroot(2)/(8)`

Exkurs: Wechsel des Wurzelverzeichnisses

- Jeder Linux-Prozess besitzt ein Wurzelverzeichnis (`/`)
  - Zugriff auf Daten außerhalb des Wurzelverzeichnisses ist **nicht** möglich
  - Kindprozesse erben das Wurzelverzeichnis ihres Elternprozesses (`fork(2)`)
- Beispiel-Code `jail.c`:

```c
int main(int argc, char *argv[])
{
  /* Starte Kindprozess (/bin/bash) nach erfolgreichem
     Wechsel des Wurzelverzeichnisses */
  if (chroot("/mnt/") == 0) {
    execl("/bin/bash", NULL);
  }

  return 0;
}
```

- Beispiel-Code `jail.c`:
- Die Datei `/mnt/bin/bash` des Live-Systems entspricht der Datei `/bin/bash` des Kindprozesses nach Aufruf von `chroot(2)`

Entwicklung eines VM-Abbilds

Systemkonfiguration

- **Hinweise**:
  - Externe Daten sind noch **vor** dem **chroot**-Wechsel mittels `scp` in das Abbild (`/mnt`) zu **kopieren**.
  - Sämtliche **Änderungen** an dem von `debootstrap` erstellten System in der `chroot`-Umgebung sind **persistent**
- Einhängen von `/proc` (manchmal notwendig, z. B. für `java`)

```
$ mount none -t proc /proc
```

- Das Skript `post-debootstrap.sh` (siehe Aufgabenstellung) beinhaltet essentielle Anpassungen für die VM-Abbild-Konfiguration
- Aufruf des Skriptes in der `chroot`-Umgebung

```
$ sh post-debootstrap.sh
Setting up /etc/apt/sources.list
(...)
Please set a password for user 'cloud'.
$ passwd cloud
```

Entwicklung eines VM-Abbilds — apt-get(8)

Software-Installation

- Ergänzen der Software des Grundsystems mittels `apt-get`
- Aktualisieren der Paketquellen (`update`) und anschließendes Einspielen potentieller Updates (`upgrade`)

```
$ apt-get update
$ apt-get upgrade
```

- Das Kommando `apt-get install` löst Abhängigkeiten auf und installiert die entsprechenden Pakete, `apt-get clean` löscht Caches

```
$ apt-get install <paket1> <paket2> ... <paketn>
$ apt-get clean
```

- Die Übung benötigt die folgenden zusätzlichen Pakete:

```
dnsutils less libc6-xen openssh-server
resolvconf screen sudo openjdk-7-jdk
```

SSH-Authentifizierung mit Schlüsseln

- SSH-Authentifizierung mit einem Schlüsselpaar **ohne** Passwort

1. Privaten und öffentlichen Schlüssel mit `ssh-keygen` **auf einem CIP-Pool-Rechner** erzeugen

```
$ ssh-keygen -f ~/<gruppen_name> -N ""
Generating public/private rsa key pair.
Your identification has been saved in <gruppen_name>.
Your public key has been saved in <gruppen_name>.pub.
(...)
```

2. Hinterlegen des **öffentlichen** Schlüssels **in chroot-Umgebung**

```
$ su - cloud
$ mkdir .ssh
$ scp <user>@<cip_pool_host>:~/<gruppen_name>.pub \
  /home/cloud/.ssh/authorized_keys
```

3. (Späterer Zugriff auf virtuelle Maschine mittels des **privaten** Schlüssels)

```
$ ssh -i ~/<gruppen_name> <vm_addr>
```

VM-Umgebung verlassen

- Shell beenden (2x ausführen)

```
$ exit
```

  - Rückkehr von Benutzerwechsel (`su - cloud`)
  - Verlassen der `chroot`-Umgebung

- Grml-Live-Umgebung herunterfahren

```
$ halt
```

  - Eingehängte Dateisysteme werden automatisch ausgehängt
  - Stellt sicher, dass alle Änderungen geschrieben wurden
  - `qemu` beendet sich

Zusammenfassung und Ausblick

- Ziel: Verlagerung der Übung in eine virtuelle Maschine

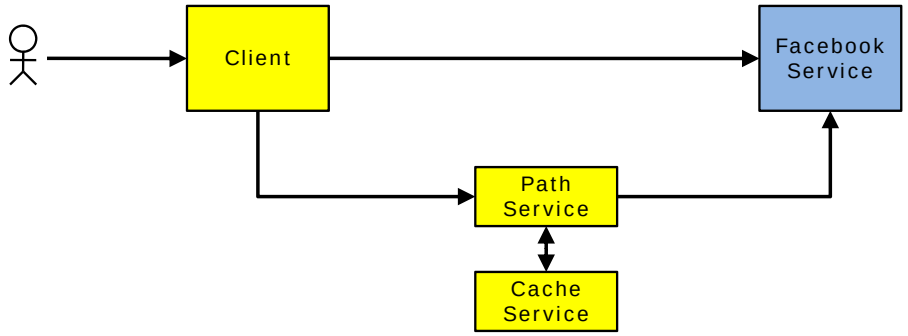
- Entwicklung des Abbilds einer virtuellen Maschine
  1. Erstellen des Containers für eine virtuelle Festplatte
  2. Erzeugen eines Dateisystems in diesem Container
  3. Verwendung eines Live-Systems für den Bootstrap-Prozess
  4. Anpassung der Konfiguration, Installation zusätzlicher Softwarepakete
  5. Hinterlegen des öffentlichen Schlüssels für die spätere Authentifizierung ohne Passwort

- Nächste Schritte
  - Verlagerung der Übungsaufgabe in eine virtuelle Maschine
  - I4-Private-Cloud-Infrastruktur des Lehrstuhls (Eucalyptus)
  - Instanziierung des Abbilds

Überblick

Aufgabe 2

- Implementierung eines Cache-Diensts
  - Verwaltung von Schlüssel-Wert-Paaren (*Objekte*)
  - Zugriff auf mehrere Objekte über einen gemeinsamen Schlüssel (*Buckets*)
- Erweiterung des Pfad-Diensts
  - Steigerung der Effizienz durch Nutzung des Cache-Diensts
  - Speicherung von Pfadberechnungen und Freundschaftsbeziehungen



Aufgabe 2

- Erzeugung und Konfiguration eines eigenen VM-Abbilds
  - Installation des Grundsystems
  - Installation von Pfad- und Cache-Dienst
- Betrieb der Dienste in der privaten Eucalyptus-Cloud des Lehrstuhls
  - Hochladen des Abbilds und Starten der virtuellen Maschine
  - Eucalyptus-Rechnerübung: Mi. 29.10., 16:00–18:00 Uhr (**s. t.**)